METRO Library Council - Fundamentals of Python 2020.02.25

Why Python?

Python is an incredibly efficient programming language and allows us to do some impressive things with only a few lines of code! Thanks to Python's syntax, we can write "clean" code that is easier to debug and allows for overall readability. Further, code written in Python is easily extendable and reusable, allowing you and others to build upon existing code. Python is used in a variety of contexts from game design to data analysis. It is also used a lot in academic research, especially in the sciences. Yet, it has utility regardless what discipline you come from or are currently working in.

Python Environment

If Python is installed on your computer, you can interact with it on the command line, sometimes referred to as the "terminal", "console", "Python shell" or "REPL" (Read, Eval, Print and Loop). More often, people use a text editor, such as [Sublime (https://www.sublimetext.com/)](https://www.sublimetext.com/), or more sophisticated IDEs such as [PyCharm (https://www.jetbrains.com/pycharm/)](https://www.jetbrains.com/pycharm/) to write and run code. With a lot of setups available, you have many options to choose from!

Today, we are using a browser-based Jupyter Notebook, which allows users to selectively run code cells and add rich text elements (paragraph, equations, figures, notes, links) in Markdown. With code, notes, instructions, and comments all in one place, it serves as a powerful resource and learning tool.

What does this lesson cover?

- Variables
- Basic Data Types
- Lists
- Dictionaries
- Loops
- Conditional Statements
- Functions and Arguments
- Python Libraries

Basic Syntax

Python is sometimes loosely referred to as 'executable pseudocode' since it often uses easily recognizable words, which can be used to infer what is happening or what will happen. Take for example, the simple line of code below. What do you think will happen when we run it?

to run press `Shift-Enter`

```
In [ ]: print("Hello, World!")
```

Variables

A variable is assigned a *value*. Once assigned, the variable holds the information associated with that value. Variables are important in programming languages. In Python, the convention is to use descriptive variables to help make clear what you are trying to do in your code. Using clear variables can help you maintain your code and can help others read and understand your code.

```
In [ ]: ## To use data, we first assign it to a "variable"

        NY_state_bird = "Eastern Bluebird"
```

You can think about it as:

> "The variable 'NY_state_bird' *gets* the value 'Eastern Bluebird'.

```
In [ ]: ## Create a variable called my_name, assign it a value, and run

        my_name = "Genevieve"
```

The value assigned to a variable will remain the same until you alter it. The value can be changed or reassigned within the program. For example, we can change the value of `message` to something new.

```
In [ ]: # Assign variable

        message = "Hello, World!"
        print(message)
```

In [ ]:

```python
# Reassign variable

message = "METRO rocks!"
print(message)
```

Why can we do this? Because Python interprets one line at a time! Be careful, once a variable has been changed, it will hold that new value. When in doubt, don't re-use variable names unless you are absolutely sure that you don't need the old value any longer.

Rules for naming variables

- can only contain letters, numbers, and underscores
- can start with an underscore, but not a number
- no spaces, but an underscore can be used to separate words or you can use CamelCase
- cannot use the names of Python built-in functions (https://docs.python.org/3/library/functions.html) or keywords (https://www.w3schools.com/python/python_ref_keywords.asp)
- should be short, but descriptive (employee_name is better than e_n)
- be consistent. If you start with CamelCase or snake_case, try to use it throughout

In [ ]:

```python
# Examples of acceptable variable names

test_1 =
employee_name =
_homework_grades =
InventoryList =
ISBN =

# Less helpful variable names

t_1 =
e_n =
_hom_grad =
inventlist =
bknr =
```

In [ ]:

```python
# keywords, built-in functions, and reserved words cannot be used as variable names

True = 3
```

In [ ]:

```python
# Spelling and syntax must be exact

message = "I'm starting to understand variables!"
print(mesage)
```

In [ ]:
```
# Create a few helpful variable names relevent to your work

GitHub_URLs =
Author_First_Name =
Author_Last_Name =
Article_Title =
Article_URLs =


# Facilitator note: have participants pair up and read each other's variables as a form of peer code review.
# Are the variables easily understood by the person reading them?
```

Data Types

There are four basic data types in Python. They are:

- string: `"Hello, World!"`
- integer: `74`
- decimal (float): `7.4`
- boolean: `TRUE` `FALSE`

In [ ]:
```
# If you are unsure what the data type is, you can use the type() method
# which returns class type of the argument(object) passed as parameter.
# the syntax would be type(variable)

message = "Hello, World"
num = 74
```

In [ ]:
```
# using the type(variable) syntax, check the data types of the variables above

type(message)
type(num)
```

Strings

A *string* is a series of unicode characters surrounded by single or double quotation marks. Strings can be anything from a short word like "is" to combinations of words and numbers like "123 Mulberry Lane," to an entire corpus of texts (all of Jane Eyre). Strings can be stored, printed, and transformed.

In [ ]:
```
# With this variable, we are printing it's type and its length with Python's built-in functions

favorite_game = "sonic the hedgehog"
print(type(favorite_game))
print(len(favorite_game))
```

In [ ]:

```
# helpful built-in functions that transform strings

print(favorite_game.title())
print(favorite_game.upper())
print(favorite_game.lower())
```

In [21]:

```
# Create a variable and assign it a string. Use two of the built-in functions and print them both.
# You should have three lines of code.

movie = "little women"
year = "(2019)"
director = "greta gerwig"

print(movie.title() + year + " directed by " + director.title() + " was pretty good!")
```

```
Little Women(2019) directed by Greta Gerwig was pretty good!
```

Concatenating Strings

Sometimes it's helpful to join strings together. A common method is to use the plus symbol (+) to add multiple strings together. Simply place a + between as many strings as you want to join together.

In [ ]:

```
# concatenate strings, in this case a first and a last name, using the + operator

first_name = "grace"
last_name = "hopper"
full_name = first_name + " " + last_name
print(full_name)
```

In [ ]:

```
# adding to this, we can use a built-in function + a longer string.

print(full_name.title() + " was a pioneer of computer programming.")
```

In [ ]:

```
# Alternatively, we could do something like this:

output = " was a pioneer of computer programming."

print(full_name.title() + output)
```

Now you try! Create two or more variables and print out a message of your own!

In [23]:

```
## Create two or more variables and use the print statement to tell us about something that interests you!
## Favorite author, actor, game, place to visit are all good options!

fav_activity = "hiking"
fav_place = "mountains"

print("My favorite thing to do is " + fav_activity + " in the " + fav_place + ".")
```

```
My favorite thing to do is hiking in the mountains.
```

Sometimes, we need need to escape characters, add spaces, line breaks, and tabs to our text.

In [ ]:

```
# Here's my example with an escape character.

first_name = "alan"
last_name = "turing"
full_name = first_name + " " + last_name
print(full_name.title() + " once said, \"Machines take me by surprise with great frequency.\"")
```

In [ ]:

```
# Adding a line break and a empty line with two consecutive "/n"

line_one = "An old silent pond..."
line_two = "A frog jumps into the pond,"
line_three = "Splash! Silence again."
Basho_Haiku = line_one + "\n" + "\n" + line_two + "\n" + "\n" + line_three
print(Basho_Haiku)
```

Challege 1

In [ ]:

```
# create a quote from a famous person and also create your own haiku (5, 7, 5)
# feel free to partner up!


#see two answers above to solve this challege
```

Tidying up Strings

Texts are messy, sometimes you need to clean them up!

Striping Whitespace

In [ ]:

```
# we can remove white space from the left and right

text = " This sentence has too many    spaces. "
print(text)

text = text.lstrip()
text = text.rstrip()
print(text)
```

Using Remove

Note: .remove( ) takes two parameters, first is what you are looking for in the text and the second is what you want to replace it with.

```
string.replace(old, new)
```

In [ ]:

```
# use replace to remove white spaces from the middle of texts, replacing
it with a single space.

text = text.replace("    ", " ")
print(text)
```

Real World Example

Working with a list of strings may require some tidying up, especially if you want to work with text as data. Often this is a process! Here's a real world example from my research on ingredients in classic recipes.

In [ ]:

```
# we can clean this text up using some of Python's built-in functions as
well as re (a regular expressions library)

import re

# Here is the in ingredient list. How do we isolate just the ingredients
by themselves?
crawfish_pasta = "1/4 cup olive oil; 5 scallions, roughly chopped; 2 clo
ves garlic, minced; 1 medium yellow onion, minced; 1/2 small green bell
 pepper, seeded and minced; 1/2 cup dry white wine; dried mint; 1 can wh
ole peeled tomatoes; 2 lb. cooked, peeled crawfish tails; 1 cup heavy cr
eam; 1/3 cup roughly chopped parsley, plus more for garnish; Tabasco; Ko
sher salt and freshly ground black pepper; 1 lb. linguine;Grated parmesa
n"
print(crawfish_pasta)
```

In [ ]:

```
# Remove some of the re-occuring text using replace

crawfish_pasta = crawfish_pasta.replace(", minced", "").replace(", rough
ly chopped", "").replace("/", "").replace(" cup", "").replace(", seeded
 and minced", "").replace("roughly chopped", "").replace(", plus more fo
r garnish", "").replace("lb. ", "")
print(crawfish_pasta)
```

In [ ]:

```
# use a simple regular expression to remove numbers and strip white spac
e

crawfish_pasta = re.sub('[0-9]', '', crawfish_pasta).strip()
print(crawfish_pasta)
```

In [ ]:

```
# replace consecutive spaces and colapse spaces following the semicolons

crawfish_pasta = crawfish_pasta.replace("   ", "").replace("; ", ";")
print(crawfish_pasta)
```

In [ ]:

```
#split at semicolon (i.e. our delimiter) and break string into a list

crawfish_pasta = crawfish_pasta.split(';')
print(crawfish_pasta)
```

In [ ]:

```
for ingredient in crawfish_pasta:
    print(ingredient)
```

In [ ]:

```
# use some of the built-in fuctions and re to clean up this messy line o
f text

messy_text = " He st0pped     before the d00r of his own cottage    , wh
ich was the fourth one from the main building and next to the 1ast. "

tidy_text = messy_text.replace("0", "o").replace("1", "l").replace("
", " ").replace("    ", "").strip()
print(tidy_text)
```

Any Questions About Strings?

Integers

An *integer* in Python is a whole number, also known as counting numbers or natural numbers. They can be positive or negative.

In [ ]:

```
3+4 # Addition
```

In [ ]:

```
4-3 # Subtraction
```

In [ ]:

```
30*10 # Multiplication
```

In [ ]:

```
30/10 # Whole division
```

In [ ]:

```
46//4 # Floor Division
```

In [ ]:

```
10**6 # An exponent operation
```

In [ ]:

```
## Order of Operations PEMDAS - Please excuse my dear Aunt Sally!
10 - ((14 / 2) * 3) + 1
```

In [ ]:

```
# determine if a number is odd or even using modulo, or remainder, operations

even_num = 10%2
odd_num = 15%2

print(even_num)
print(odd_num)
```

Why? An even number has no remainder when divided by 2. An odd number will have a remainder.

Floats

A *float* in Python is any number with a decimal point.

In [ ]:

```
# volume of a rectangle

height = 3.5
length = 7.25
width = 7.75
V = height*length*width
print(V)
```

In [ ]:

```
# using int means the value is restricted to the non-decimal number

int(V)
```

In [ ]:

```
# the round function helps handle floating point numbers (floats).
# the number after the comma (in this case 2) sets the precision of the decimal.
# this example restricts the numbers after the decimal point to 2 (handy when printing sums that refer to money)

round(V, 2)
```

In [ ]:

```
# If you need only what is after the decimal point, you could do something like this:

dec_portion = V - int(V)
print(dec_portion)
```

Avoiding TypeError with the Str() Function

We've talked about strings and we've talked about Integers. When using them in the same space, we need to avoid type errors.

In [ ]:

```
# strings and integers are different data types:

fav_num = 3.14
message = "My favorite number is " + fav_num + "."
print(message)
```

In [ ]:

```
# The str() function is used to convert the specified value, in this case an integer, into a string.

message = "My favorite number is " + str(fav_num) + "."
print(message)
```

Booleans

A *boolean* is a binary variable, having two possible values called “true” and “false.”.

In [ ]:

```
3>=4
```

In [ ]:

```
3 == 4
```

In [ ]:

```
x = 255
y = 33

if x > y:
  print("x is greater than y.")
else:
  print("x is not greater than y.")
```

Challenge 2

In [ ]:

```
# Working with a partner, we are going to create some helpful conversions and print them.
# hint: begin with identifying the information you need and assign it to a variable

# 1: Convert farenheit to celsius:
# formula: celsius = (farenheit - 32) * 5/9

farenheit = 77
celsius = (farenheit - 32) * 5/9
print(celsius)


# 2: Convert celsius to farenheit:
# formula: farenheit = (celsius * 9/5) + 32
celsius = 25
farenheit = (celsius * 9/5) + 32
print(farenheit)


# 3: Convert pounds to kilograms:
# formula: kilograms = pounds/2.2046226218

pounds = 105
kilograms = pounds/2.2046226218
print(round(kilograms))


# 4: Convert kilograms to pounds:
# formula: pounds = kilograms * 2.2046226218

kilograms = 48
pounds = kilograms * 2.2046226218
print(round(pounds))
```

-------BREAK----------

Lists

A *list* is a collection of items in a particular order. Lists can contain strings, integers, floats and each element is separtated by a comma. Lists are always inside square brackets.

In [ ]:

```
# An example of a list

my_grocery_list = ["bananas", "pears", "apples", "chocolate", "coffee", "cereal", "kale", "juice"]
print(my_grocery_list)
```

In [ ]:

```
# we can use the len function to see how many items are in a list

len(my_grocery_list)
```

In [ ]:

```
# remember, Python indexing starts at zero

print(my_grocery_list[2])
```

In [ ]:

```
# using the end index will give you the last item in the list

print(my_grocery_list[-1])
```

In [ ]:

```
# If you need a range in a list, you can use can set a range using start_index:end_index

print(my_grocery_list[3:5])
```

In [ ]:

```
# Sorting will put your list in alphabetical order

print(sorted(my_grocery_list))
```

In [ ]:

```
# using the sorted fuction is not permanent

print(my_grocery_list)
```

In [ ]:

```
# to make it so, use the sort function

my_grocery_list.sort()
print(my_grocery_list)
```

In [ ]:

```
# we can also permanently reverse this list

my_grocery_list.reverse()
print(my_grocery_list)
```

Dictionaries

A *dictionary* in Python is a collection of *key-value pairs*. Each key is connected to a value and you can use a key to access the value associated with that key.

In [ ]:

```
#dictionaries can hold lots of information
#uses a key, value pair

club_member_1 = {
    "member_name": "Brian Jones",
    "member_since": "2017",
    "member_handle": "@bjones_tweets",
}
print(club_member_1)
```

In [ ]:

```
#to get information from the dictionary

for key, value in club_member_1.items():
    print(key, ":", value)
```

In [ ]:

```
# we can nest dictionaries into a list

club_member_2 = {
    "member_name": "Anne Green",
    "member_since": "2015",
    "member_handle": "@Greenbean",
}

club_members = [club_member_1, club_member_2]
print(club_members)
```

In [ ]:

```
for member in club_members:
    for key, value in member.items():
        print(key, ":", value)
    print()
```

Challenge 3

In [ ]:

```python
# Make a list of 7 items
# print the list
# reverse and print the list
# print the middle three elements of the list
# OPTIONAL: use the member information above to create a club_member_3, append it to club_members,
# and print the new list. HINT: remember, codes is reusable.

list = ["ALA", "ASIS&T", "MLA", "ARLIS/NA", "ACRL", "PLA", "METRO"]
print(list)
print()
list.reverse()
print(list)
print()
print(list[2:4])
print()

club_member_3 = {
    "member_name": "Jane Smith",
    "member_since": "2010",
    "member_handle": "@JS_Designs",
}

club_members.append(club_member_3)

for member in club_members:
    for key, value in member.items():
        print(key, ":", value)
    print()
```

List Manipulation: Adding, Editing, and Removing Items in a list

In [ ]:

```python
# this is a list of items in our makerspace

makerspace = ["Arduino", "3D printer", "sewing machine", "sewing machine case", "soldering iron", "micro controlers", "Legos", "Legos case"]
print(makerspace)
```

In [ ]:

```python
# we can add items to our list using append function
# appending adds these items to the end of the list

makerspace.append("camera")
makerspace.append("SD cards")
makerspace.append("batteries")
print(makerspace)
```

```
In [ ]: # If we want to insert at a particular place, can can use the insert fuction

        makerspace.insert(3, "thread") # insert takes two parameters: list.insert.(index, element)
        print(makerspace)
```

```
In [ ]: # we can also use an index to replace an existing element with a new element

        makerspace[0] = "Raspberry Pi"
        print(makerspace)
```

```
In [ ]: # the del function deletes at an index
        # del can be used in lists, but not strings

        del makerspace[4]
        print(makerspace)
```

```
In [ ]: # the pop function also removes elements from a list at a specific index

        makerspace.pop(7)
        print(makerspace)
```

Challenge 4

```
In [ ]: # append "record player" to the end of the list, insert "gramophone" at index 2,
        # delete "CD" from the list, reverse and print.

        analog_media = ["VCR", "tape player", "16 mm projector", "CD", "reel to reel", "8 track"]

        analog_media.append("record player")
        analog_media.insert(2, "gramaphone")
        del analog_media[4]
        print(analog_media)
```

For Loops

Looping allows you to take the same action(s) with every item in a list. This is very useful for when you have lists that contain alot of items.

```
In [ ]: # range starts at the first number and goes up to the number BEFORE the stopping number

        for number in range(1,6):
            print(number)
```

In [ ]:

```
# range can also include negative numbers

for new_number in range(-3,4):
    print(new_number)
```

In [ ]:

```
# for each number, square the number

for number in range(1,6):
    print(number,"squared =", number**2)
```

In [ ]:

```
# use an index counter to track the index of each name

name_list = ["Denise", "Tammy", "Belinda", "Byron", "Keelie", "Charles",
"Alison", "Jamal", "Greta", "Manuel"]
name_index = 0
for name in name_list:
    print(name_index, ":", name)
    name_index += 1
    # incrementing by one
```

In [ ]:

```
musicians = ['Allen Toussaint', 'Buddy Bolden', 'Danny Barker', 'Dr. Joh
n', 'Fats Domino', 'Irma Thomas']
for musician in musicians:
    print(musician)
```

In [ ]:

```
musicians = ['Allen Toussaint', 'Buddy Bolden', 'Danny Barker', 'Dr. Joh
n', 'Fats Domino', 'Irma Thomas']
for musician in musicians:
    print("I have a record by "+ musician + ".")
```

Conditional Statements

IF Statements

An *if statement allows* you to check for a condition(s) and take action based on that condition(s). Logically it means:

```
if conditional_met:
    do something
```

In [ ]:

```
# test to see if a number is greater than or equal to a particular value

age = 18
if age >= 18:
    print("You can vote!")
```

In [ ]:

```
# loop through list and print each name in it

name_list = ["Kylie", "Allie", "Sherman", "Deborah", "Kyran", "Shawna",
"Diane", "Josephine", "Peabody", "Ferb", "Wylie"]
for name in name_list:
    print(name)
```

In [ ]:

```
# loop through list and use if to find a particular name

for name in name_list:
    if name == "Diane":
        print(name, "found!")
```

Else Statements

In [ ]:

```
# Diane has RSVP'd to say that she can't make it and we should take her
 off the list

counter = 0
for name in name_list:
    if name == "Diane":
        print(name, "found! Located at index: ", counter)
        break
    else:
        print("not found ...", counter)
        counter += 1
```

In [ ]:

```
# Locating the exact index makes it easier to remove her name

del name_list[6]
print(name_list)
```

In [ ]:

```
# Python has built-in finding functions that make this very easy

name_list2 = ["Kylie", "Allie", "Sherman", "Deborah", "Kyran", "Shawna",
"Diane", "Josephine", "Peabody", "Ferb", "Wylie"]
rsvp_yes = ["Kylie", "Allie", "Shawna","Josephine", "Peabody", "Ferb",
"Wylie"]
rsvp_no = ["Kyran","Sherman","Diane"]

for name in name_list2:
    if name in rsvp_no:
        print(name, "RSVP'd no, removing from list ...")
        name_list2.remove(name)
    else:
        if name in rsvp_yes:
            print(name, "RSVP'd yes and is coming to the pary!")

print()
print(name_list2)
```

Elif Statement

In [ ]:

```
# separate the elements into other lists by type

my_list = [9, "Endymion", 1, "Rex", 65.4, "Zulu", 30, 9.87, "Orpheus", 1
6.45]
my_int_list = []
my_float_list = []
my_string_list = []

for value in my_list:
        if(type(value)==int):
                my_int_list.append(value)
        elif(type(value)==float):
                my_float_list.append(value)
        elif(type(value)==str):
                my_string_list.append(value)

print(my_list)
print(my_int_list)
print(my_float_list)
print(my_string_list)
```

OPTIONAL: If, Elif, Else Statements

In [ ]:

```
# counting number ranges

#create a list of 50 random numbers between 1 and 100
import random
number_range_list = []
for entry in range(0,50):
    number=random.randint(1,100)
    number_range_list.append(number)

# count and categorize numbers by range
first_quarter = []
second_quarter = []
third_quarter = []
fourth_quarter = []

# check ranges
for number in number_range_list:
    #print(number)
  if number <= 25:
      first_quarter.append(number)
  elif number <=50:
      second_quarter.append(number)
  elif number <=75:
      third_quarter.append(number)
  else:
      fourth_quarter.append(number)

# calculate percentage of whole in each quarter
q1_total = round((len(first_quarter)/50)*100)
q2_total = round((len(second_quarter)/50)*100)
q3_total = round((len(third_quarter)/50)*100)
q4_total = round((len(fourth_quarter)/50)*100)

print("data ready ...")
```

In [ ]:

```
# print out the data to give a basic shape to it visually

print("Quick, General Bar-Chart \n")
print("  0-25:",first_quarter)
print(" 26-50:",second_quarter)
print(" 51-75:",third_quarter)
print("76-100:",fourth_quarter)
```

In [ ]:

```
# print out the analysis

print("Distribution of 50 randomly generated numbers \n by percentage pe
r qarter:")
print("  0-25:",q1_total, "%")
print(" 26-50:",q2_total, "%")
print(" 51-75:",q3_total, "%")
print("76-100:",q4_total, "%")
```

In [ ]:

```
# break down the information further

print("Minimum Values")
print("  0-25:",min(first_quarter))
print(" 26-50:",min(second_quarter))
print(" 51-75:",min(third_quarter))
print("76-100:",min(fourth_quarter))
```

In [ ]:

```
print("Maximum Values")
print("  0-25:",max(first_quarter))
print(" 26-50:",max(second_quarter))
print(" 51-75:",max(third_quarter))
print("76-100:",max(fourth_quarter))
```

In [ ]:

```
print("Average Values")
print("  0-25:",round(sum(first_quarter)/len(first_quarter)))
print(" 26-50:",round(sum(second_quarter)/len(second_quarter)))
print(" 51-75:",round(sum(third_quarter)/len(third_quarter)))
print("76-100:",round(sum(fourth_quarter)/len(fourth_quarter)))
```

While Statements

A *while satement* operates as long as a particular condition is true. Logically it means:

```
while conditional_true:
    do something
```

In [ ]:

```
# what do you think this will do?
counter = 10;
while counter>0:
    print("Counter =", counter)
    #print("Counter = " + str(counter))
    counter = counter-1

#What do you think would happen if at the end of our code we wrote "counter = counter + 1"?
```

Infinite Loops occur when a condition is never met. This can cause the program to run out of memory and crash.

Functions

Function 1

```
In [ ]: # data for function 1

        famous_programmers = ["augusta ada king (lovelace)", "grace hopper", "li
        mor fried", "evelyn boyd granville",
        "parisa tabriz", "sister mary kenneth keller", "carol shaw"]
```

```
In [ ]: # create a function by using the keywork 'def'
        # set the argument(s) it will accept inside the parentheses
        # remember, a function must be defined before it is called

        def thank_you(names_list):
            for name in names_list:
                message = "Thank you " + name.title() + ". Your work and contrib
        utions to programming have been amazing!"
                print(message)
```

```
In [ ]: # call the function and use the list as the argument

        thank_you(famous_programmers)
```

Function 2

```
In [ ]: # data for function 2

        weekly_high_temp = [44, 56, 47, 50, 53, 57, 61]
```

```
In [ ]: # building on the challenge you did previously let's create a function t
        hat does a conversion
        # instead of a single value, let's take in a list and convert it

        def farenheit_to_celsius(temperature_list):
            temp_in_celcius = []
            for number in temperature_list:
                celcius = round((number - 32) * 5/9)
                temp_in_celcius.append(celcius)
            return temp_in_celcius
```

```
In [ ]: print("Weekly High Temps in Celcius:", farenheit_to_celsius(weekly_high_
        temp))
```

Challenge 5

In [ ]:

```
# Create a function to convert this list of feet to inches


measurements_feet = [7, 9, 14, 33, 18.5, 45, 3.25, 10, 1, 2.5]

def feet_to_inches(feet):
    measurement_inches = []
    for foot in feet:
        inches = foot * 12
        measurement_inches.append(inches)

    return measurement_inches

print(feet_to_inches(measurements_feet))
```

Python Libraries

Python has a huge collection of libraries, also known as packages, which are essentially a collection of modules in a directory. Python is a package itself and comes with many built in modules. There are, however, many other packages that are useful if you need to complete certain tasks, such as data cleaning, web scrapping, or statistical analysis.

Since you already have Anaconda, use Anaconda package repository (https://anaconda.org/anaconda/repo) to install python libraries you need!

Web Scraping with BeautifulSoup

install BeautifulSoup on the terminal using Anconda's repo for bs4 (https://anaconda.org/anaconda/beautifulsoup4)

```
conda install -c anaconda beautifulsoup4
```

In [ ]:

```
from bs4 import BeautifulSoup
import requests
import time

url = "http://shakespeare.mit.edu/Poetry/sonnets.html"

results_page = requests.get(url)
page_html = results_page.text
soup = BeautifulSoup(page_html, "html.parser")


sonnets = soup.find_all('dl')

for sonnet in sonnets:

        each_sonnet = sonnet.find_all('a')

        for each in each_sonnet:

                title = each.text

                url = each['href']
                url = "http://shakespeare.mit.edu/Poetry/" + url

                sonnet_request = requests.get(url)
                sonnet_html = sonnet_request.text
                sonnet_soup = BeautifulSoup(sonnet_html, "html.parser")

                sonnet_text = sonnet_soup.find("blockquote")
                sonnet_text = sonnet_text.text


                print(title)
                print()
                print(url)
                print()
                print(sonnet_text)
                print("---------------------")


                time.sleep(2)
```